Weekly Work Report

Wenjie Niu

VISION@OUC

July 22, 2018

1 Content

This week, I start the learning of shallow neural network. The main stucture is the following.

1.1 Neural Network Presentation

Like Figure. 1, the neural network is figure. 2. There are the input features $x_1x_2x_3x_4$ stacked vertically, which is called the input layer of the neural network. In the middle, this is called hidden layer of the neural network, but the final layer here is called output layer, which is responsible for generating the predicted value $\hat{y}$. The term hidden layer refers to the fact that in a training set the values for these nodes in the middle are not observed that you don't see what they should be in the training set. The other denotion of input features is $a^{[0]}$ and the term a also stands for activitions refering to the values that different layers of the neural network are passing on to the subsequent layers. So the input layer passes on the value x to the hidden layer. The hidden layer and the output layers will have parameters, the hidden layer will have associated with their parameters w and b. Similarly, the output layer has associated with parameters $w^{[2]}$ and $b^{[2]}$.

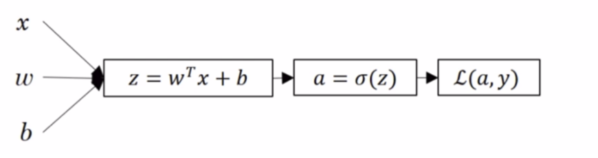


Figure 1: Logistic Regression [1, 2]

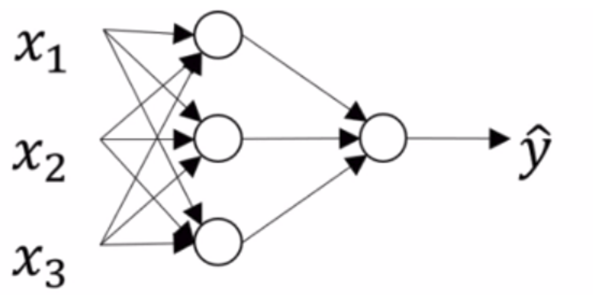


Figure 2: Neural Network Model [1, 2]

1.2 Computing a Neural Network

In the figure. 3, the circle images the regression really represent two steps of computation. First compute z as follows and in second compute the activation as a sigmoid function of z. Neural network just does this a lot more times. Vectorizing the equtions in figure. 4, it becomes to

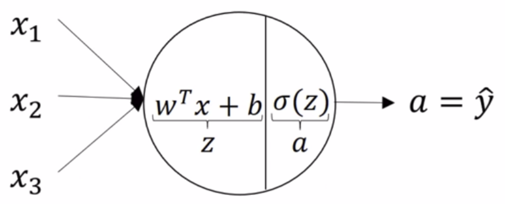
$$Z^{[1]} = W^{[1]}X + b^{[1]} \tag{1}$$

$$a^{[1]} = \sigma(Z^{[1]}) \tag{2}$$

When having a nueral network who have one hidden layer, we need to compute the four equations in figure. 5, and it can be seen as a vectorized implementation of computing the output of first four logistic regression units in a hidden layer.

1.3 Explaination for vectorized implementation

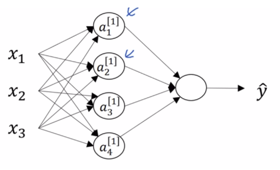
The figure. 6 justifies why $Z^{[1]} = W^{[1]}X + b^{[1]}$. That's a correct vectorization of the first step of the four steps.



$$z = w^T x + b$$

$$a = \sigma(z)$$

Figure 3: Logistic Regression Detail [1, 2]



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \; a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \; a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \; a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \; a_4^{[1]} = \sigma(z_4^{[1]})$$

Figure 4: Neural Network Representation [1, 2]

Given input x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

Figure 5: Neural Network Representation Learning [1, 2]

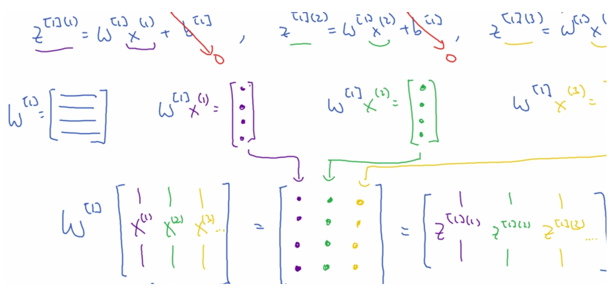


Figure 6: Justification for vectorized implementation [1, 2]

1.4 Activation functions

In the forward propagation steps for the neural network we have these two steps where we use σ function, so that σ is called an activation function as shown int figure. 7. In the more general case we ca have a different function $g(z)$ where g could be a nonlinear function that may not be the σ function. The tanh function or the hyperbolic tangent function is as shown in figure. 7. Using σ it makes sense for $\hat{y}$ to be a number that thr output is between 0 and 1 rather than between -1 and 1. The one exception where the σ activation function is used is when using binary classification in which case you might use the σ activation function for the output layer. The activation functions can be different for different layers. Now one of the downsides of both the σ function and the tanh function is that if z is either very large or very small then the gradient of the derivative or the slope of this function becomes very small being close to 0. Another popular function is rectified linear unit(ReLU) and the ReLU function looks like as shown in figure. 7 and the formula is $a = max(0, z)$. So the derivative is 1 so long as z is positive and derivative or the slope is 0 when z is negative. While the derivative when z is exactly 0 is not well-defined, but you could pretend a derivative either 1 or 0 when $z = 0$. One disadvantage of the ReLU is that the derivative is equal to 0 when z is negative. There is another version of the ReLU called the leaky ReLU as shown in figure. 7, instead of it being 0 when x is negative. It takes a slight slope like so this is called the Leaky ReLU.

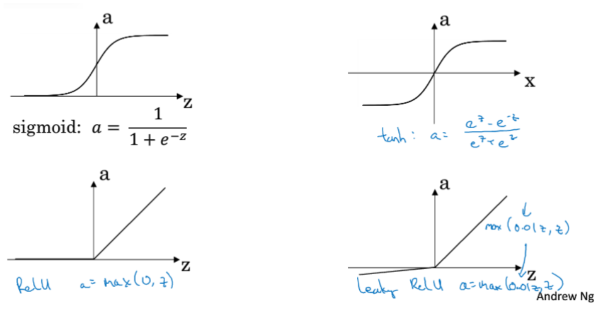


Figure 7: Pros and cons of activation functions [1, 2]

1.5 Derivatives of Activation functions

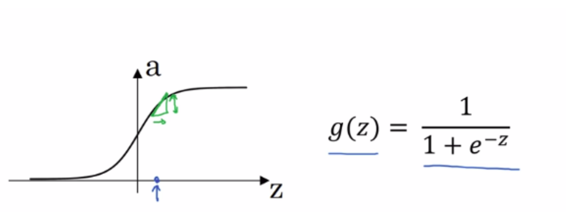


Figure 8: Sigmoid Activation Function [1, 2]

As shown in figure. 8 then the slope of the function is

$$\frac{d}{dz}g(z) = g(z)(1 - g(z)) \tag{3}$$

First if z is very large, then $g(z)$ will be close to 1,the equation. 3 is equal to 0 and the slope is close to 0. Conversely, if z is very small, then $g(z)$ is close to 0 and the equation. 3 will be close to 0. Finally at $z = 0$ then $g(z)$ is euqal to $\frac{1}{2}$ so the derivative is equal to $\frac{1}{4}$.

As shown in figure. 9 then the slope of the function is

$$\frac{d}{dz}g(z) = 1 - (tanh(z))^2 \tag{4}$$

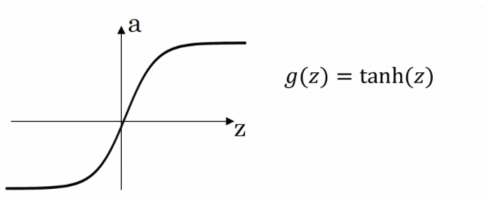


Figure 9: Tanh Activation Function [1, 2]

If $z = 10$, then $tanh(z) = 1$ while $g^{'}(z) = 0$; If $z = -10$, then $tanh = -1$ while $g^{'}(z) = 0$; If $z = 0$, then $tanh =$ while $g^{'}(z) = 1$;.

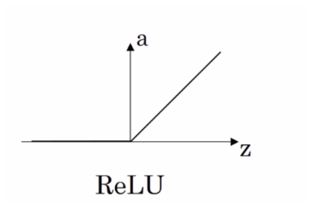


Figure 10: ReLU Activation Function [1, 2]

As shown in figure. 10 then the slope of the function is

$$g^{'}(z) = \begin{cases} 0, & if \quad z < 0 \\ 1, & if \quad z > 0 \\ undefined, & if \quad z = 0. \end{cases} \tag{5}$$

It doesn't matter you could set the derivative to be equal to when $z = 0$.

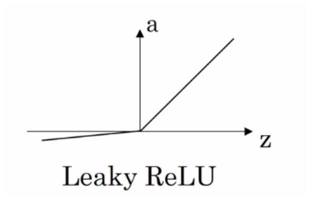


Figure 11: Leaky ReLU Activation Function [1, 2]

As shown in figure. 11 $g(z) = max(0.0iz, z)$ then the slope of the function is

$$g^{'}(z) = \begin{cases} 0.01, & if \quad z < 0 \\ 1, & if \quad z > 0 \\ undefined, & if \quad z = 0. \end{cases} \tag{6}$$

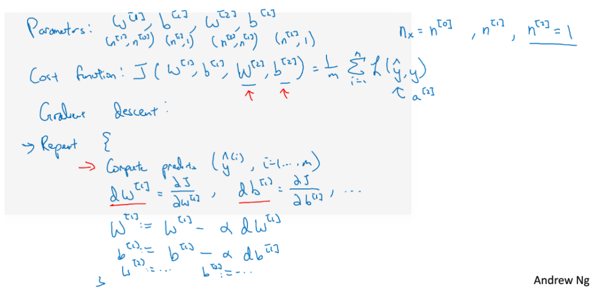
1.6 Gradient Descent for Neural Networks

The figure. 12 is one iteration of gradient descent and then you are repeating this some number of times until your parameters look like they're converging.

There are forward and backward propagation equations in figure. 13.

1.7 Random Initialization

Technically I'm assuming that the outging weights and bias are identical to 0. If initializing the neural network this way, the hidden units are completely indentical so they're completely symmetric



Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$

$(n^{[1]}, n^{[0]})$ $(n^{[1]}, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[2]}, 1)$

$n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1$

Cost function: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$ $\leftarrow a^{[2]}$

Gradient descent:

Repeat {

Compute predicts $(\hat{y}^{(i)}, i = 1 \dots m)$

$dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$, ...

$W^{[1]} := W^{[1]} - \alpha \, dW^{[1]}$

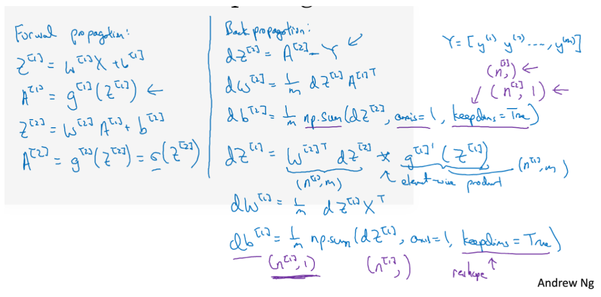
$b^{[1]} := b^{[1]} - \alpha \, db^{[1]}$

$W^{[2]} := \dots$ $b^{[2]} := \dots$

}

Andrew Ng

Figure 12: Gradient Descent for Neural Networks [1, 2]



Forward propagation:

$Z^{[1]} = W^{[1]} X + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$

$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$

Back propagation:

$dZ^{[2]} = A^{[2]} - Y$

$Y = [y^{(1)} \ y^{(2)} \dots, y^{(m)}]$

$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$

$db^{[2]} = \frac{1}{m}$ np.sum$(dZ^{[2]}$, axis=1, keepdims=True)

$(n^{[2]},)$ $(n^{[2]}, 1)$

$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]}, m)}$

element-wise product

$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$

$db^{[1]} = \frac{1}{m}$ np.sum$(dZ^{[1]}$, axis=1, keepdims=True)

$(n^{[1]}, 1)$ $(n^{[1]},)$ reshape

Andrew Ng

Figure 13: Formulas for Computing Derivatives [1, 2]

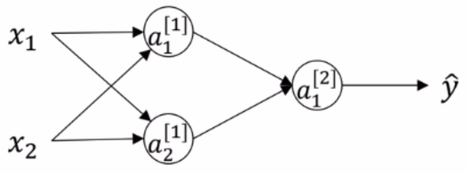


Figure 14: A Neural Network [1, 2]

which means that the computing exactly the same function. Then after one iteration that the same statement is still true, the two hidden units are still symmetric. So in this case there's really no point to having more than one hidden unit. The solution to this is to initialize your parameters randomly. We can set $W^{[1]} = np.randome.randn((2,2))$*0.01 so initialize it to very small random values. Initialize $b^{[1]} = np.zeros((2,1))$ so long as W is initialized randomly. Then similarly for $W^{[2]}$ and so on.

1.8 Deep L-layer Neural Network

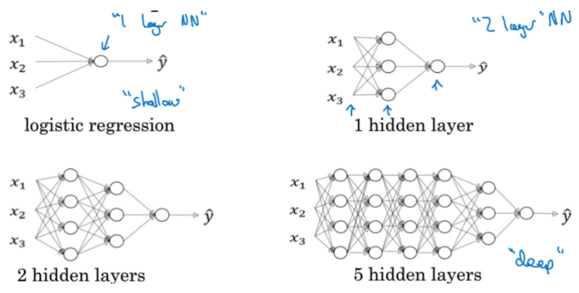


Figure 15: Neural Networks [1, 2]

Shallow versus depth is a matter of degree. Technically logistic regression is a one layer neural network. The shallow and deep neural network is as shown in figure, 15.

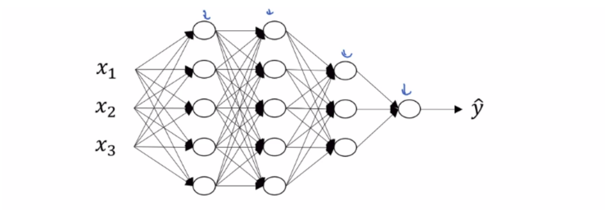


Figure 16: Deep Neural Network Notation [1, 2]

Here is a four layer neural network with 3 hidden layers as shown in figure. 16 and the number fo units in these hidden layers are 5,5,3 then there's one output unit.

1.9 Forward Propagation in a Neural Network

Given a single training example x here's how you can compute the activations of the first layer so for the first layer we compute $Z^{[1]} = W^{[1]}X + b^{[1]}$ which $W^{[1]}$ and $b^{[1]}$ are parameters that affect the activations in layer 1 of neural network and then we compute the activations for $a^{[1]} = g^{[1]}(Z^{[1]})$. So if you do thatt you've now computed the activations from layer 1. Similarly to layer 2 and so on. X here is equals to $a^{[0]}$, in a word $z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$, $a^{[l]} = g^{[l](z^{[l]})}$. That's the general forward propagation equations. After vectorization, the equations become $Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$, $A^{[l]} = g^{[l](Z^{[l]})}$

1.10 Getting Your Matrix Dimensions Right

If you meet the neural network like figure. 17, the general formula to check $W^{[l]}$ is that when we're implementing the metrix for a layer l to the dimension of that matrix will be $(n^{[l]}, n^{[l-1]})$ while $b^{[l]}$ is $(n^{[l]}, 1)$. If implementing back propagation then the deimention of dW should be the same as dimension of W so dW is $(n^{[l]}, n^{[l-1]})$ and db should be the same dimension as b so db is $(n^{[l]}, 1)$.

When forward propagation, the dimension of $Z^{[1]}$ and $a^{[1]}$ is $(n^{[l]}, 1)$, after vectorized is $(n^{[l]}, m)$. When implementing backward propagation, the dimensions is the same as forward propagation. So make sure that all the matrices dimensions are consistent.

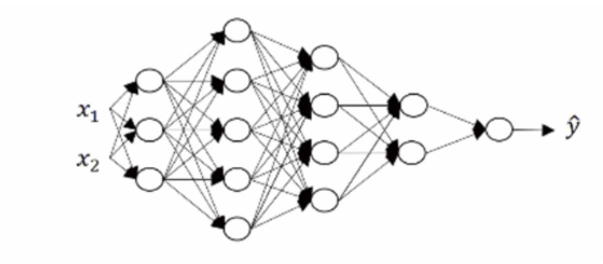
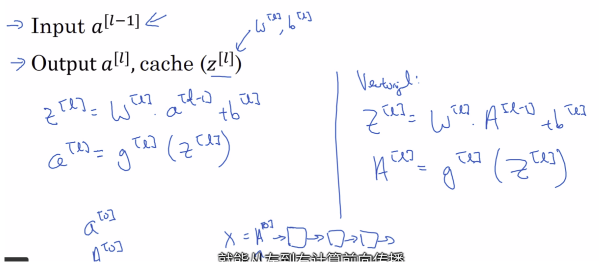


Figure 17: Decide parameters $W^{[l]}$ and $b^{[l]}$ [1, 2]

1.11 Forward and backward Propagation



→ Input $a^{[l-1]}$

→ Output $a^{[l]}$, cache ($z^{[l]}$) $w^{[l]}, b^{[l]}$

$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

Vectorized:

$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$

$A^{[l]} = g^{[l]}(Z^{[l]})$

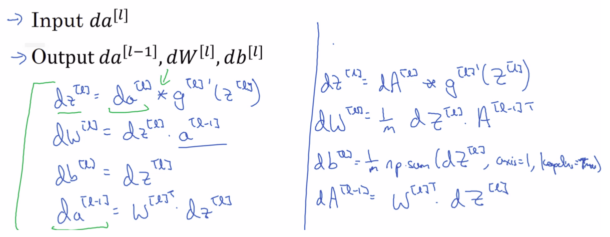
$a^{[0]}$

$A^{[0]}$

$X = A^{[0]} \to \square \to \square \to \square \to$

Figure 18: Forward Propagation for Layer l [1, 2]

The procedure of forward propagation is in figure. 18 from left to right. The procedure of backward propagation is in figure. 19.



→ Input $da^{[l]}$

→ Output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$

$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]}$

$db^{[l]} = dz^{[l]}$

$da^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$

$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$

$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$

$db^{[l]} = \frac{1}{m}$ np.sum($dZ^{[l]}$, axis=1, keepdims=True)

$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$

Figure 19: Backward Propagation for Layer l [1, 2]

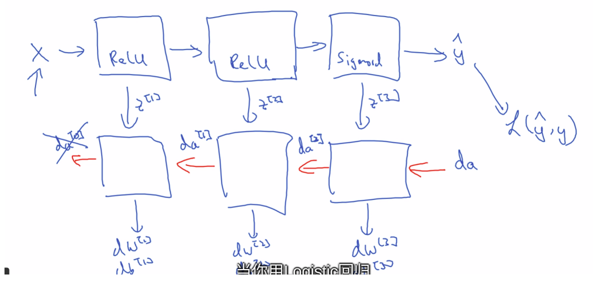
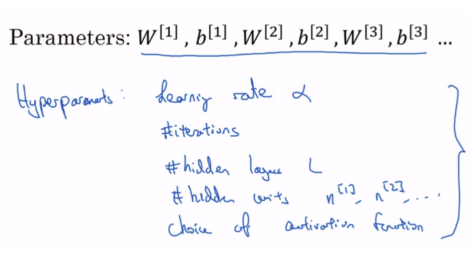


Figure 20: Summarize the procedure [1, 2]

Summarize all the procedure of forward and backward propagation is in figure. 20.

1.12 Parameters vs Hyperparameters

Being effective in developing your deep neural nets requires that not only orgnized parameters well but also hyperparameters.



Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$ ...

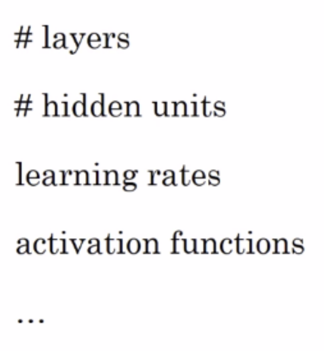
Hyperparameters: learning rate α
#iterations
#hidden layer L
#hidden units $n^{[1]}, n^{[2]}, \ldots$
choice of activation function

Figure 21: Parameters and Hyperparameters [1, 2]

All of the things are that you need to tell the learning algorithm and so these parameters that control the ultimate parameters W and b so we call all of these things below hyper parameters. Because it is the hyper parameters that somehow determine the final value of the parameters W and b that we end up with. In fact, deep learning has a lot of different hyper parameters like Momentum, minibatch size and so on.

1.13 Train/dev/test sets

From now, the part 2 course begins. We'll learn the pratical aspects of how to make neural network work well ranging from things like hyperparameter tuning to how to set up data to how to make sure optimization algorithm runs quickly so that get the learning algorithm to learn in a reasonable time. In the first week, it about the cellular machine learning problem, then about randomization and about some tricks for making neural network implementation is correct. When training a neural network we have to make a lot of decisions such as how many layers will your neural network have and so on as shown in figure. 22.



layers

hidden units

learning rates

activation functions

...

Figure 22: Many decisions should be made [1, 2]

2 Progress in this week

Step 1 Watched the courses clips.

Step 2 Wathced again and took notes.

Step 3 Grasped the related pictures and wrote the Latex.

Step 4 Organized the content and push to the github.

3 Plan

Objective: Learn Neural Network and Deep Learning by myself.

2018.07.22—2018.07.28	Watch Part 2 course clips and take the note.
2018.07.29—2018.08.04	Do so on week four course.
2018.08.05—2018.08.11	Do so on next part course.

References

[1] A. Ng. Neural network and deep learning. `http://mooc.study.163.com/smartSpec/detail/1001319001.htm`.

[2] A. Ng. Neural network and deep learning. `https://www.coursera.org/learn/neural-networks-deep-learning/exam/QR8kq/introduction-to-deep-learning`.